Empfohlene Unterrichtsinhalte als Vorbereitung für dieses Thema: 01-02 (Was ist Python?), 02-01 (Variablen), 02-02 (Strings), 02-03 (Listen), 03-01 (Indexing und Slicing), 03-02 (Bedingungen)

Einführung in die computerlinguistische Programmierung mit Python

03-03: `for`: Schleifen für Sequenzen 🔃

Wir haben beim Indexing und Slicing (Thema 03-01) gesehen, dass man mit einem Index ein einzelnes Element aus einer Sequenz auswählen kann. Oft wollen wir aber **alle Elemente nacheinander** auswählen und bestimmte Operationen auf sie anwenden.

Wenn die Sequenz wenige Elemente hat, können wir das manuell machen:

In [ ]:

```
sequenz = "1234"

print(sequenz[0])
print(sequenz[1])
print(sequenz[2])
print(sequenz[3])
```

Unsere Programme sollen aber so geschrieben sein, dass die **Logik nicht verändert** werden muss, wenn die Variablenwerte anders gesetzt werden. Deshalb ist es besser, wenn wir ein neues Konstrukt verwenden: die `for`-Schleife. Damit können wir Vorgänge, die auf jedes Element einer Sequenz angewendet werden, **ein einziges Mal definieren** - wie oft die Schleife dann ausgeführt wird, darum kümmert sich der Interpreter, abhängig davon, wie viele Elemente in der Sequenz enthalten sind.

Alle Befehle, die in der Schleife stehen, werden schrittweise **immer wieder für jedes einzelne Element der Liste** ausgeführt. Nach dem letzten Element wird die Schleife verlassen und die Befehle unterhalb der Schleife werden ausgeführt.

Unser Beispiel von oben sieht dann so aus:

In [ ]:

```
sequenz = "1234"

for c in sequenz:
    print(c)
```

Wir haben hier eine neue Variable namens `c` angelegt, die bei jedem Schleifendurchlauf den Wert des jeweils aktuellen Elements annimmt. `c` hat hier also nacheinander die Werte `"1"`, `"2"`, `"3"`, und `"4"`.

`for`-Schleifen entsprechen immer dem folgenden Muster:

```
for <element> in <collection>:
    <befehl1>
    <befehl2> # (optional)
    <befehl3> # (optional)
    <...>
```

Die Zeile, die mit `for` beginnt und mit `:` endet, heißt **Kopf** der Schleife. Die Zeilen, die darunter stehen und weiter **eingerückt** sind als der Kopf, heißen **Körper**. Befehle, die im Körper einer Schleife stehen, müssen immer gleich weit eingerückt sein (typischerweise 4 Leerzeichen pro Einrückungsebene).

Auf Englisch heißt Einrückung **indentation** (als Verb: **to indent**).

Im Beispiel oben enthielt der Körper der Schleife nur einen Befehl. Hier noch ein Beispiel, in dem mehrere Codezeilen zur Schleife gehören, erkennbar an der Einrückung:

In [ ]:

```
# Liste definieren und Inhalt in einer Variable speichern
zahlen_bis_fuenf = [0, 1, 2, 3, 4, 5]

print("Start")    # Das Wort "Start" ausgeben

# Diese Zeile leitet die Schleife ein.
for zahl in zahlen_bis_fuenf:
    # Erster Befehl in der Schleife:
    print("Aktuelle Zahl:")
    # Zweiter Befehl in der Schleife:
    print(zahl)

# Befehle ab hier gehören nicht mehr zur Schleife.

print("Ende")      # Das Wort "Ende" ausgeben
```

Befehle, die die gleiche Einrückung haben wie der Kopf der Schleife, werden außerhalb der Schleife ausgeführt, also nur einmal und nur vor/nach der Ausführung der Schleife.

Es ist dabei egal, ob die `collection` (beispielsweise eine Liste) als Variable übergeben wird oder als konkreter Wert. Wir können das Beispiel von oben also folgendermaßen umschreiben, ohne die Funktionalität des Programms zu verändern:

In [ ]:

```
#####################################
# +++ erste Lösung (siehe oben) +++ #
#####################################
zahlen_bis_fuenf = [0, 1, 2, 3, 4, 5]

print("Start")    # Das Wort "Start" ausgeben

# Diese Zeile leitet die Schleife ein.
for zahl in zahlen_bis_fuenf:
    # Erster Befehl in der Schleife:
    print("Aktuelle Zahl:")
    # Zweiter Befehl in der Schleife:
    print(zahl)

# Befehle ab hier gehören nicht mehr zur Schleife.

print("Ende")      # Das Wort "Ende" ausgeben
```

In [ ]:

```
#########################################
# +++ zweite Lösung (keine Variable) +++ #
#########################################

print("Start")     # Das Wort "Start" ausgeben

# Diese Zeile leitet die Schleife ein.
for zahl in [1, 2, 3, 4, 5]:
    # Erster Befehl in der Schleife
    print("Aktuelle Zahl:")
    # Zweiter Befehl in der Schleife
    print(zahl)

print("Ende")      # Das Wort "Ende" ausgeben
```

Das `element` bekommt einen Namen, den wir im Kopf der Schleife angeben. Im Beispiel oben lautet der Name `zahl` . Das jeweils aktuelle Element ist dann im Körper der Schleife unter diesem Variablennamen abrufbar. Deshalb wird im ersten Schleifendurchlauf die Zahl 1 ausgegeben, im zweiten die Zahl 2 usw.

Nachdem die Schleife **beendet** ist, behält die Variable den letzten zugewiesenen Wert.

Tipp: Da der Name des Elements sich pro Schleifendurchlauf auf genau ein Element bezieht, wählen wir typischerweise einen Namen im **Singular** (wie `zahl`). Der Name der Variable, in der die Liste gespeichert ist, steht hingegen im Plural (wie `zahlen_bis_fuenf`). Durch diese Konvention wird Code gut lesbar, weil Kollektionen (wie Listen) und individuelle Elemente auf den ersten Blick unterschieden werden können.

Fortlaufende Listen von Zahlen kann Python übrigens mit dem Befehl `range()` für uns erzeugen, ohne dass wir jede Zahl aufschreiben müssen - so wie im nächsten Codebeispiel. Das ist nützlich, wenn wir eine lange Liste brauchen (z.B. Zahlen bis 100) oder die Anzahl der Schleifendurchläufe durch andere Werte im Programm bestimmt werden soll.

Eine weitere Benennungskonvention ist, dass wir in Schleifen wie der hier folgenden das aktuelle Element mit dem Buchstaben `i` bezeichnen. Das `i` steht für *Integer* oder *Index*.

In [ ]:

```
# Welcher Wert versteckt sich hinter range(10)?
for i in range(10):
    print(i)
```

In [ ]:

```
# range([min], max) erzeugt eine Kollektion, die alle Ganzzahlen
# zwischen min (inklusive; falls nicht angegeben, 0) und max (exklusive) enthält.
# Probieren wir es aus!
for i in range(2,5):
    print(i)
```

Es gibt zwei klassische Muster, nach denen wir mit Schleifen durch Sequenzen **iterieren** können. Das erste gibt uns pro Schleifendurchlauf den Wert des aktuellen Elements:

In [ ]:

```
for c in "word":
    print(c)
```

Die zweite Methode ist etwas komplexer: Hier wird nicht direkt über die Sequenz iteriert, sondern über alle Zahlen, die als Index in der Sequenz vorkommen. Das erreichen wir über eine Kombination von `range()` und `len()`:

In [ ]:

```
unser_text = "voll langer text"
for i in range(len(unser_text)):
    print(i)
    print(unser_text[i])
```

Wie man sieht, enthält diese Art von Schleife etwas **mehr Informationen**. Oft reicht es aus, nur mit den Elementen zu iterieren (erste Variante). Die Indizes sind immer dann interessant, wenn wir z.B. das aktuelle Element mit dem Element direkt davor oder direkt danach vergleichen wollen. Dann können wir anhand der Laufvariable `i` auch auf andere Indizes zugreifen, indem wir bspw. `unser_text[i-1]` auswählen.

Variablenwerte in Schleifen überschreiben mit `+` oder `+=`

Wir können Variablenwerte innerhalb von Schleifen überschreiben. Zum Beispiel können wir eine Zählvariable nach und nach erhöhen:

In [ ]:

```
lower_case_counter = 0

for w in ["Im", "traurigen", "Monat", "November", "war's"]:
    # Prüfen, ob das aktuelle Wort kleingeschrieben ist
    if w.islower():
        lower_case_counter = lower_case_counter + 1

print("Kleingeschriebene Wörter: " + str(lower_case_counter))
```

In diesem Beispiel haben wir über mehrere Schleifendurchläufe mehrfach den aktuellen Wert der Variable abgefragt, 1 dazu addiert, und das Ergebnis dieser Operation in die gleiche Variable geschrieben, um die Zahl zu erhöhen. Das funktioniert, weil der Interpreter immer zuerst die **rechte Seite einer Variablenzuweisung vollständig auswertet**. Es gibt keinen Konflikt, wenn eine Variable sowohl links als auch rechts vom Zuweisungsoperator auftaucht.

Python bietet eine Kurzschreibweise für `a = a + b` an: Wir schreiben dann `a += b`. Das Beispiel von oben würde dann folgendermaßen aussehen:

In [ ]:

```
lower_case_counter = 0

for w in ["Im", "traurigen", "Monat", "November", "war's"]:
    # Prüfen, ob das aktuelle Wort kleingeschrieben ist
    if w.islower():
        lower_case_counter += 1

print("Kleingeschriebene Wörter: " + str(lower_case_counter))
```

Auch die Operatoren `-=`, `*=` und `/=` stehen zur Verfügung.

Achtung! Die Reihenfolge der Zeichen spielt eine Rolle! Wenn wir `=+` schreiben statt `+=`, wird das Programm sich nicht so verhalten wie vorgesehen. Könnt ihr euch vorstellen, warum?

Zusammenfassung

- Schleifen ermöglichen es uns, beliebigen Code automatisch auf alle Elemente einer Sequenz nacheinander anzuwenden.
- Im Kopf einer Schleife vergeben wir eine selbstgewählte Laufvariable mit einem beliebigen Namen. Diese Variable enthält uns im Körper der Schleife immer den Wert des gerade aktuellen Elements.
- Jede eingerückte Zeile unter einem Schleifenkopf gehört zur Schleife. Durch die Einrückung weiß der Interpreter, welche Befehle mehrfach ausgeführt werden sollen und welche nur einmal ausgeführt werden.
- Die Schleife wird immer genau so oft ausgeführt, bis alle Elemente in der Sequenz einmal behandelt wurden.
- Mit `range()` können wir durch Zahlensequenzen mit beliebigem Start und Ende iterieren.
- Mit `range(len(...))` können wir durch alle vorhandenen Indizes in einer Sequenz iterieren.

Weitere Themen dieser Woche

- 03-01: Indexing und Slicing
- 03-02: Bedingungen
- 03-04: Zufallszahlen